

DATA STRUCTURE

**All Notes Are Verified by NPTEL and Published
By**

WWW.ASKTOHOW.COM

Introduction:

- In many applications it is necessary to order give objects as per an attribute. For example, arranging a list of student information in increasing order of their roll numbers or arranging a set of words in alphabetical order.
- The attribute by which objects are arranged or sorted is called key. In the first example roll number is the key.
- One class of sorting methods are based on comparisons. Most important methods in this class are Insertion, Sink, Selection, Quick, Merge, and Heap sort algorithms. We have already discussed Insertion sort algorithms in the module 2. We discuss some of these algorithms in this module.

Sink Sort

- Main idea in this method is to compare two adjacent elements and put them in proper order if they are not.
- Do this by scanning the element from first to last.
- After first scan, largest element is placed at last position. This is like a largest object sinking to bottom first.
- After second, scan second largest is placed at second last position.
- After n passes all elements are placed in their correct positions, hence sorted.

Input	Pass 1	Pass 2	Pass 3	Pass 4	Pass 5	Pass 6	Pass 7	Pass 8
12	12	12	12	12	12	12	12	12
32	18	18	18	18	18	14	14	14
18	24	24	19	19	14	18	18	18
24	30	19	24	14	19	19	19	19
30	19	28	14	24	24	24	24	24
19	28	14	28	28	28	28	28	28
28	14	30	30	30	30	30	30	30
14	32	32	32	32	32	32	32	32

[An Applet Demonstration Of Sink Sort](#)

- The algorithmic description of the Sink sort is given below:

Algorithm Sink-Sort (a[n])

Step 1 :

`for i = 0 to n-2 do`

Step 2 : `for j = 0 to n-i do`

Step 3 : `if (a[j] > a[j+1]) then`

Step 4 : `swap(a[j], a[j+1]);`

- The algorithm can be modified, such that, after each pass next smallest element reach to the top position. This is like a bubble coming to top. Hence called **Bubble-Sort**.

- As we have discussed earlier, purpose of sorting is to arrange a given set of records in order by specified key. In the case of student records, key can be roll number. In the case of employees records, key can be employ identification number. That is, sorting a set of records based on specific key.
- Insertion and Sink sorts compare keys of two records and swap them if the keys are not in order. This is not just swapping keys, we have to swap whole records associated with those keys.
- The time required to swap records is proportional to its size, that is number of fields.
- In these methods, same record may be swapped many times, before reaching its final position in sorted order.
- This method, selection sort, minimized the number of swaps. Hence efficient if the size of the records are large.
- Look at keys of all records to find a record with smallest key and place the record in the first place.
- Next, find the smallest key record among remaining records and swap with the record at second position.
- Continue this procedure till all records are placed correctly.

[An Applet Demonstration of Selection Sort](#)

	12	32	18	24	30	19	28	14
After Pass 1	12	32	18	24	30	19	28	14
Pass 2	12	14	18	24	30	19	28	32
Pass 3	12	14	18	24	30	19	28	32
Pass 4	12	14	18	19	30	24	28	32
Pass 5	12	14	18	19	24	30	28	32
Pass 6	12	14	18	19	24	28	30	32
Pass 7	12	14	18	19	24	28	30	32

- Pseudo code of the algorithm is given below.

Algorithm Selection_Sort ($a[n]$)

```

Step 1:   for i = 0 to n-2 do
Step 2:   lowest-key = i
Step 3:   for j = i+1 to n-1 do {
Step 4:       if (a[i].key > a[j].key) then
Step 5:           lowest_key = j
Step 6:       swap(a[i], a[lowest_key]); }

```

- The time complexity of the sorting algorithms discussed till now are in $O(n^2)$. That is, the number of comparisons performed by these algorithms are bound above by cn^2 , for some constant $c > 1$.
- Can we have better sorting algorithms? Yes, merge sort method sorts given a set of number in $O(n \log n)$ time.
- Before discussing merge sort, we need to understand what is the meaning of merging?

Merge sort Method :

Definition: Given two sorted arrays, $a[p]$ and $b[q]$, create one sorted array of size $p + q$ of the elements of $a[p]$ and $a[q]$.

- Assume that we have to keep $p+q$ sorted numbers in an array $c[p+q]$.
- One way of doing this is by copying elements of $a[p]$ and $b[q]$ into $c[p+q]$ and sort $c[p+q]$ which has time complexity of atleast $O(n \log n)$.
- Another efficient method is by merging which is of time complexity of $O(n)$.
- Method is simple, take one number from each array a and b , place the smallest element in the array c . Take the next elements and repeat the procedure till all $p+q$ element are placed in the array c .

[An Applet Demonstration of Merging](#)

Pseudocode of the merging is given below.

Algorithm Merge (a[p], b[q])

Step 1: i = 0

Step 2: j = 0

Step 3: k = 0

Step 4: while (i < p) && (j < q) do{

Step 5: if (a[i] < b[j]) {

Step 6: c[k] = a[i] ;

Step 7: i = i + 1 ; }

Step 8: else {

Step 9: c[k] = b[j]

Step 10: j = j + 1 }

Step 11: k = k + 1 }

Step 12: if (i == p) then

Step 13: while (j < q) do{

Step 14: c[k] = b[j] ;

Step 15: k = k + 1 ;

Step 16: j = j + 1 ;

 }

Step 17: else

Step 18: while (i < p) do

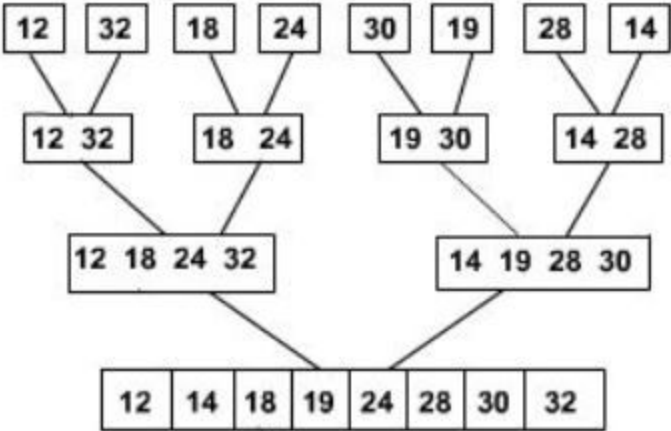
Step 19: { c[k] = a[i] ;

Step 20: k = k + 1 ;

Step 21: i = i + 1 ;
 }

We can assume each element in a given array as a sorted sub array. Take adjacent arrays and merge to obtain a sorted array of two elements. Next step, take adjacent sorted arrays, of size two, in pair and merge them to get a sorted array of four elements. Repeat the step until whole array is sorted.

Following figure illustrates the procedure.



- Assume that procedure `Merge1(a, i, j, k)` takes two sorted arrays `a[i..j]` and `a[j+1..k]` as an input and puts their merged output in the array `a[i..k]`. That is, in the same locations. Pseudocode of the merge sort is given below.

Algorithm `Merge-Sort` (`a`, `p`, `q`)

```

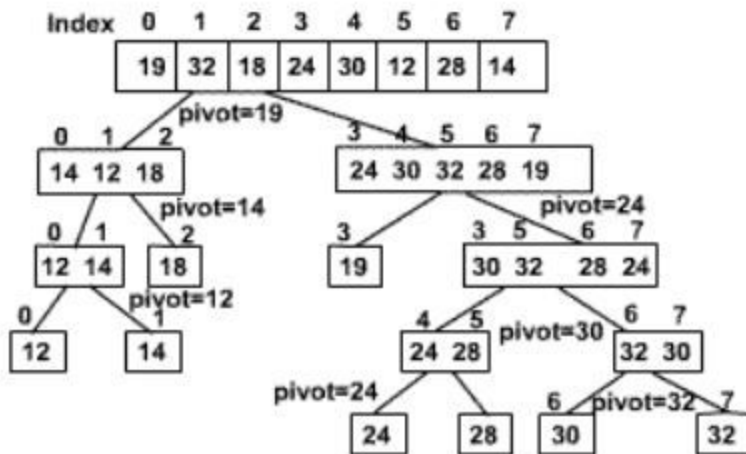
Step 1:  if (p < q) {
Step 2:      mid = (p+q)/2 ;
Step 3:      Merge-Sort(a, p, mid) ;
Step 4:      Merge-Sort(a, mid+1, q) ;
Step 5:      Merge(a, p, mid, q) ;

          }

```

- The above procedure sorts the elements in the sub array `a[p..q]`.
- To sort given any `a[n]` invoke the algorithm with parameters `(a, 0, n - 1)`.

- Another very interesting sorting algorithm is **Quick sort**. Its worst time complexity is $O(n^2)$, but it is a faster than many optimal algorithms for many input instances.
- It is an another divide and conquer algorithm.
- Main idea of the algorithm is to partition the given elements into two sets such that the smaller numbers into one set and larger number into another. This partition is done with respect to an element called pivot. Repeat the process for both the sets until each partition contains only one element.



Pseudocode of the partition is given below.

Algorithm `partition` (`a`, `p`, `q`)

Step 1: `pivot = a[p];`

Step 2: `i = p-1 ; j=q+1;`

Step 3: `while (i<j)`

Step 5: `repeat`

Step 6: `i = i + 1`

Step 7: `until a[i] >= pivot;`

Step 8: `repeat`

Step 9: `j = j - 1`

Step 10: `until a[j] <= pivot;`

Step 11: `if (i < j) then`

Step 12: `swap (a[i], a[j]);`

Step 13: `return j`

Algorithm `quick-sort` (`a`, `p`, `q`)

Step 1: `if (p < q) {`

Step 2: `mid = partition(a,p,q);`

Step 3: `quick-sort (a, p, mid);`

Step 4: `quick-sort (a, mid+1, q);`
 `}`

- All the algorithms discussed till now are comparison based algorithms. That is, comparison is the key for sorting.
- Assume that the input number are three decimal digits. First we sort the numbers using least significant digit. Then by next significant digit and so on.
- See following example for illustration.

327	470	418	146
476	382	327	173
285	173	146	259
418	285	259	285
568	476	568	327
382	146	470	382
146	327	173	418
259	418	476	470
173	568	382	476
470	259	285	568
Input	Sorted by LS digit	Sorted by Middle digit	Sorted by Most Sig digit

[An Applet Demonstration of Radix Sort](#)

- Next question is how to sort with respect to a least significant digit or any significant digit?
- We can use any sorting algorithm we have discussed.
- Another way by maintaining a BIN for each digit from 0 to 9. If the digit is X put the number in BIN X. Concatenate the BINs from 0 to 9 to get a sorted list of numbers of that significant digit.

- The method is given in the following pseudo code

Algorithm radix-sort (a)

```
Step 1:   for i = 0 to 9 do
Step 2:     empty-bin(i);
Step 3:     for position = least significant digit to most significant
              digit
Step 4:       for i = 1 to n do
Step 5:         x = digit in the position of a[i];

Step 6:         put a[i] in BIN x;
Step 7:         i =1;
Step 8:         for j = 0 to 9 do
Step 9:           while (BIN(j) <> empty) do
Step 10:            a[i] = get-element(BIN(j));
Step 11:            i = i + 1;
```

- Procedure `get-element(bin)` get the next element from the bin.
- we can use Queues for implementing Bins.

Problems:

1. Write a procedure for the merge procedure Merge1(a, i, j, k).
2. Write non-recursive procedure for merge sort.
3. Modify merge sort procedure to sort number in non-ascending order.
4. For what input, merge sort takes maximum number of comparisons.
5. Implement merge sort algorithm in c.
6. Trace the Quick sort algorithm on sorted array of elements 1 to 10.
7. Trace the Quick sort algorithm on elements 10, 9, 8, 1.
8. Write a program for quick sort method.
9. Implement the radix sort.
10. Design an algorithm to sort given names using radix sort.